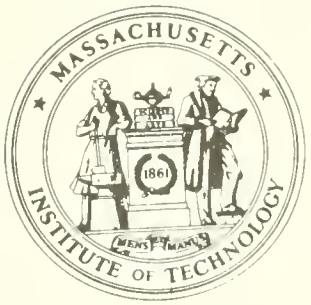


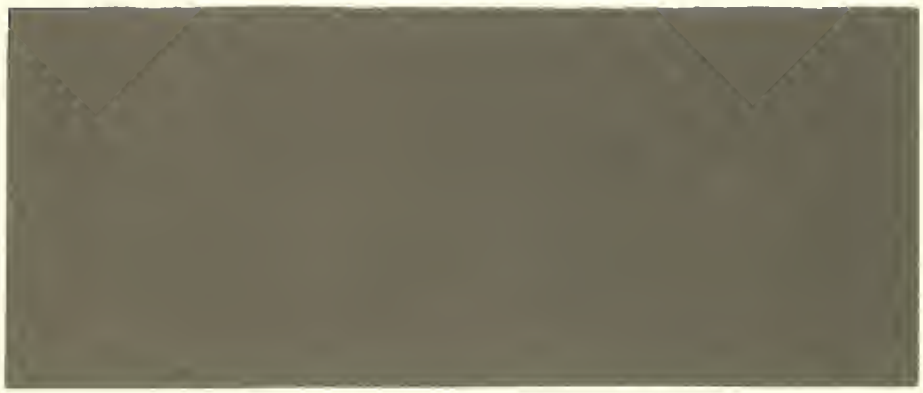
WORKING PAPER
ALFRED P. SLOAN SCHOOL OF MANAGEMENT

BRANCH AND BOUND METHODS FOR COMBINATORIAL PROBLEMS

178-66

John D. C. Little

MASSACHUSETTS
INSTITUTE OF TECHNOLOGY
50 MEMORIAL DRIVE
CAMBRIDGE, MASSACHUSETTS 02139



BRANCH AND BOUND METHODS

FOR COMBINATORIAL PROBLEMS

178-66

John D. C. Little

March 25, 1966

Paper prepared for presentation at the TAPPI Operations Research Symposium, Philadelphia, March 28-30, 1966.

Abstract

The use of branch and bound methods to solve certain types of combinatorial problems is discussed. The approach is illustrated on the traveling salesman problem. A 6 city problem is worked out in detail. Then a flow diagram is given for a general discrete variable minimization problem. Finally, applications in job sequencing, flow-shop scheduling, and traffic light synchronization are described.

Branch and bound methods have been found useful for solving certain rather difficult optimization problems. Two notable examples are the traveling salesman problem[12] and the three machine scheduling problem[14,9]. The branch and bound idea offers a general framework for the optimization rather than a complete specification. Detailed procedures must be worked out to fit each problem type and these are an important determinant of the computational success of the method. However, the successes so far offer encouragement that new problems can be approached the same way.

We shall illustrate the process on the traveling salesman problem. The branch and bound approach will then be discussed in general and a flow diagram given for it. Finally some further applications will be described.

1. The Traveling Salesman Problem

The traveling salesman problem owes its fame to looking easy but being hard. The problem may be stated as follows: A salesman starting in one city wishes to visit each of n-1 other cities and return to the start. In what order should he visit the cities to minimize the distance traveled? For "distance" we can substitute time, cost, or other criterion as desired. In fact, the most interesting current applications have nothing to do with salesmen but, as we shall see, are concerned with scheduling a sequence of jobs on a machine. This application is of considerable interest to the present audience.

The difficulty of the problem is entirely computational since the number of "tours" (i.e. feasible solutions) is finite. Enumeration of all tours, however, would be a discouraging process for a problem of any appreciable size. There are (n-1)! tours. Doubling the size of a problem from 5 to 10 cities multiples the number of tours by about 15 thousand; doubling from 10 to 20 cities, by about 250 billion.

To put the branch and bound algorithm in perspective, we report briefly on other approaches to the traveling salesman problem. Several methods have been developed for finding a good but not necessarily optimal tour. One of these is random search. Since any permutation of the first n integers can be interpreted as a tour, it is an easy matter to generate a tour by a random process, evaluate its cost, and compare it to the best of any previously developed solutions. Since these steps can be performed quickly, a large number of tours can be generated, and the best one will usually be a good solution. Furthermore, the statistics generated make it possible to say something about the probability of finding a better solution. This type of approach has been investigated by Heller [8] for a somewhat different problem and in much more detail with more sophisticated random search procedures by Reiter [16,17] for the traveling salesman problem. On a somewhat similar tack Karg and Thompson [10] have developed a heuristic program. However, their program is designed primarily for the symmetric case. Such approximate methods are important, especially for large problems. However, it is obviously desirable to find solutions that are optimal

and known to be so, whenever this can be achieved with a reasonable amount of computation.

Gilmore and Gomory [4] have an elegant, efficient algorithm for a case of traveling salesman problems where the cost matrix is of a special type. In addition, some specific problems [1] have been solved by tailor-made arguments developed for the particular numbers involved.

Consider then methods that (1) guarantee optimality (2) seem reasonable to program and, (3) are general. One method is to formulate the problem as an integer program [15,18]. However, to date I know of no published computational results. Held and Karp [7] and Gonzalez [5] have formulated the problem by dynamic programming and have coded it for the computer. The largest problem solved has been 13 cities. Beyond this, time and storage requirements go up very rapidly. Finally, we have the branch and bound methods developed by Murty, Sweeney, Karel, and myself [12]. Problems vary in difficulty for this method. Our prinicpal class of test problems has been asymmetric cost matrices constructed from 3 digit random numbers. We have solved problems up to 40 cities in size. Symmetric problems taken off maps are harder but a 25 city problem has been solved without difficulty. For details, see [12]. We have solved no sequencing problems but I know of specific instances in which others have solved a 17 city and a 21 city problem based on live data. Thus, although it is not hard to construct large problems that cannot be solved in a reasonable length of time and it

would probably be possible to construct some especially difficult small problems, it seems fair to say that problems up to, say, 20 cities, appear to be in quite good shape. Furthermore, as we shall point out, the method can be programmed so that, if the algorithm is stopped short of completion, a feasible solution is produced that is usually rather good and in some cases is optimal. More generally, some of the approaches used here might be useful in a heuristic program for the problem.

We shall simultaneously present the algorithm and work out an example. Verbal arguments will be given at each step. Detailed proofs will be omitted.

Consider the matrix of Figure 1. The entry at position (i,j) say c(i,j) represents the cost (distance) for going from city i to city j. A tour is a set of city pairs, e.g.,

$$t = [\ (1,3)\ (3,2)\ (2,5)\ (5,6)\ (6,4)\ (4,1)\ ]$$

which spell out a trip that goes to each city once and only once. Let z be the cost of a tour. From Figure 1 it may be seen that the above tour would cost:

$$z = 43 + 13 + 30 + 5 + 9 + 21 = 121.$$

If a constant is subtracted from each element of the first row of Figure 1, that constant is subtracted from the cost of every tour. This is because every tour must include one and only one element from the first row. The relative costs of all tours, however, are unchanged and so the tour that would be optimal is unchanged. The same argument can be applied to the columns.

TO

FROM	1	2	3	4	5	6
1	∞	27	43	16	30	26
2	7	∞	16	1	30	25
3	20	13	∞	35	5	0
4	21	16	25	∞	18	18
5	12	46	27	48	∞	5
6	23	5	5	9	5	∞

Figure 1. Cost matrix for a 6-city Problem

	1	2	3	4	5	6
1	∞	11	27	(10) 0	14	10
2	1	∞	15	(1) 0	29	24
3	15	13	∞	35	5	(5) 0
4	(1) 0	(0) 0	9	∞	2	2
5	2	41	22	43	∞	(2) 0
6	13	(0) 0	(9) 0	4	(2) 0	∞

Figure 2. Cost matrix after row and column reduction

The process of subtracting the smallest element of a row from each element of a row will be called reducing the row. Thus the first row in Figure 1 can be reduced by 16. Note that, in terms of the unreduced matrix, every trip out of city 1 (and therefore every tour) will have a cost of at least 16. Thus, the amount of the reduction constitutes a lower bound on the length of all tours in the original matrix.

Step 1: Reduce the rows and columns of the cost matrix. Save the sum of the reductions as a lower bound on the cost of a tour.

The results of reducing Figure 1 are shown in Figure 2. The total reduction is 48 and so $z \geq 48$ for all tours.

Next we split the set of all tours into two disjoint subsets. This is conveniently indicated by drawing a tree as in Figure 3. The node (branching point) containing "all tours" is self-explanatory. The node containing 1,4 represents all tours which include the city pair (1,4). The node containing $\overline{1,4}$ represents all tours which do not. From the 1,4 node we might later want to branch again, say, on the basis of (2,1). In Figure 3 the node containing $\overline{2,1}$ represents all tours which include both (1,4) and (2,1). In general, by tracing back from a node to the start we can pick up which city pairs are specified to be in and which out of the tours represented by the node. If the branching process is carried far enough, some node will eventually represent a single tour. Notice that at any stage of the process, the union of the sets represented by the terminal nodes is the set of all tours.

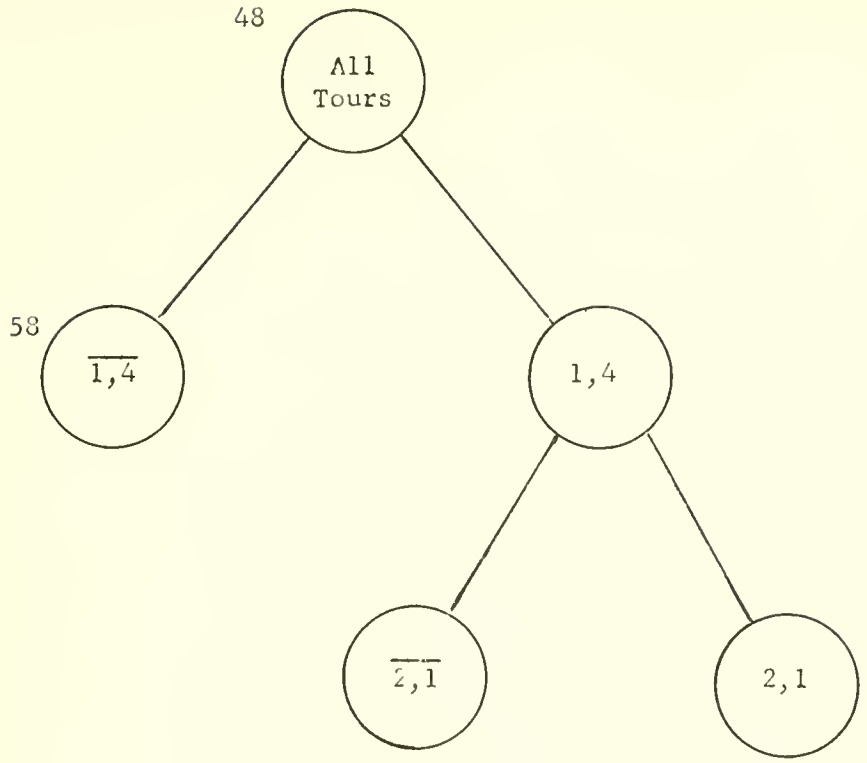


Figure 3. Start of tree

	1	2	3	4	5	6
~~1~~	~~∞~~	~~11~~	~~27~~	~~0~~	~~14~~	~~10~~
2	0	∞	14	0	28	23
3	15	13	∞	35	5	0
4	∞	0	9	∞	2	2
5	2	41	22	43	∞	0
6	13	0	0	4	0	∞

Figure 4. Matrix after deletion of row 1 and column 4

Returning to the question of lower bounds, we have seen that 48 is a lower bound on all tours. In Figure 2 the starting node is accordingly marked with 48. Consider the first row of Figure 2. Suppose that city pair (1,4) is not in a tour. Then since city 1 must be assigned to some city, the tour must incur a cost of at least 10 (the second smallest element in row 1). This is on top of the 48 already reduced out of the matrix. Similarly, since some city must be assigned to city 4, the tour must further incur the cost of the second smallest element in column 4. The element happens to be zero in this case and so the total lower bound is still 58. The node has been so marked in Figure 3.

Thinking ahead, suppose we eventually discover a tour with $z = 56$. Then, in a search for the optimal tour, it would be unnecessary to examine any of the tours in the node $\overline{1,4}$ since they all have $z \geq 58$. Not knowing in advance what z's will be found, we shall select the element on which to branch so as to increase most this lower bound. Let $\theta(k,\ell)$ be the jump in lower bound if (k,ℓ) is chosen for branching. As indicated in the example:

$\theta(k,\ell)$ = smallest element in row k, omitting the (k,ℓ) element
+ smallest element in column, omitting the (k,ℓ) element

Thus we have:

Step 2: Given a node, X, from which to branch next, and given the cost matrix associated with X, find the city pair (k,ℓ) which maximizes θ. Extend the tree from X to a node $\overline{k,\ell}$. Add $\theta(k,\ell)$ to the lower bound of X to set the lower bound of the new node.

Inspection of the matrix will show that $\theta(k,\ell) = 0$ unless $c(k,\ell) = 0$ so that the search for max θ is confined to the zeros of the matrix. In Figure 2 the θ values are shown in small circles. The largest occurs for (1,4), as already chosen to illustrate branching.

Next we put in a step needed later to detect when the end of a tour is near.

Step 3: If the number of city pairs committed to the tours of X is n-2, go to Step 6. Otherwise continue.

Returning to the example, we seek a lower bound for the 1,4 node. Since all tours here contain (1,4), row 1 and column 4 are no longer needed and may be crossed out. Next observe that no tour in the set represented by this node can contain the city pair (4,1). This is because (1,4) (4,1) constitute a two city subtour, and it is impossible for a tour to contain a subtour. Therefore, without loss of tours, set $c(4,1) = \infty$. This will prevent (4,1) from being chosen as part of a possible tour. After the changes in the matrix, further reduction of rows or columns may be possible. In the case at hand, row 2 can be reduced by one. See Figure 4. Adding this to the previous lower bound gives 49, which has been marked on the node in Figure 5. Summarizing:

Step 4: Finish the branching based on (k,ℓ) by extending the tree from X to a node $k,\ell;\ \overline{m,p}$. Here (m,p) is the city pair which would join the ends of the longest connected path involving (k,ℓ) in the set of committed city pairs of the new node. Delete row k and column ℓ in the cost matrix and set $c(m,p) = \infty$. Reduce rows and columns if possible and

add the amount of the reduction to the lower bound of X to set the lower bound for the new node.

A further example of finding (m,p) is illustrated by the node $\overline{2,1}$. All tours at this node contain (2,1) and (1,4). The addition of (4,2) would create a 3 city subtour, which we shall prevent by specifying $c(4,2) = \infty$ for the node. One could also set $c(1,2) = \infty$ but row 1 has already been crossed out and so there is no advantage. The only worthwhile candidate for (m,p) is always the element that completes the longest subtour involving (k,ℓ).

We now repeat the basic branching procedure of Steps 2 and 4; extending the tree, and working up the lower bounds as we go. Thus:

Step 5: If no node has been found which contains only a single tour, find the terminal node with the smallest lower bound and return to Step 2 for branching. Otherwise continue.

A comment is required about branching to the right versus branching to the left. Branching on the k,ℓ node (to the right in Figure 5) is the usual operation. It involves crossing out rows and columns and other manipulations which are conveniently done on the same physical representation of the matrix, whether stored on paper or in a computer. When one returns to extend a $\overline{k,\ell}$ node (moving to the left) it is usually advantageous to reconstruct an appropriate matrix from the original cost matrix, rather than arranging to save cost matrices for each node as it is laid out. We, therefore, give:

Step 2a: If a cost matrix is needed for node X, start with the original cost matrix and

(1) Find the city pairs committed to be in the tours of X and add their cost elements together to form part of the lower bound for X.

(2) For every (i,j) which is one of these city pairs, cross out the i^{th} row and j^{th} column of the matrix. Add infinities at prohibited city pairs.

(3) Reduce the remaining matrix and add the amount of the reductions to that found in (1) to give a lower bound for X.

(4) Perform the instructions given in Step 2.

The lower bound and the reduced matrix are not necessarily unique, consequently, when a new bound is calculated, the old is discarded. However, bounds calculated in the two ways will usually be the same.

As Steps 1-5 are performed on the example, the tree of Figure 5 is developed, up to the final node: 4,3; 6,2.

Here, Step 3 causes a jump out of the loop. Step 2 will have told us that the current branching is based on (4,3). This, plus the city pairs found by tracing back the tree, show that n-1 city pairs are already committed to the upcoming node. But the n-1 determine what the last must be (here (6,2)), and we have found a single tour node. Examination of the final 2X2 cost matrix will show that no reduction is possible and that the cost of the remaining pairs relative to this matrix is zero. Thus the tour has z = 63.

Furthermore, since every possible tour is contained in one or another of the terminal nodes and each of the other nodes has a lower bound greater than 63, we have found the optimal tour.

Step 6: If entry to this step is from Step 3, the next node is k,ℓ; m,p where (m,p) is the only city pair left after crossing out row k and column ℓ. The node contains a single tour. Calculate its cost. For any entry to the step, let t_0 denote the tour with the smallest cost, say z_0, among single tours found so far. If z_0 is less than or equal to the lower bound of every terminal node on the tree, t_0 is optimal. Otherwise choose the multiple tour terminal node with the smallest lower bound and return to Step 2 for branching.

At this point, let us stand back and review the general motivation of the algorithm. It proceeds by branching, crossing out a row and column, blocking a subtour, reducing the cost matrix to set a lower bound and then repeating. Although it is fairly clear that the optimal solution will eventually be found, why should these particular steps be expected to be efficient? First of all, the reduction procedure is an efficient way of building up lower bounds and also of evoking likely city pairs to put into the tour. Branching is done so as to maximize the lower bound on the $\overline{k,\ell}$ node without worrying about the k,ℓ node. The reasoning here is that the k,ℓ node represents a smaller problem, one with the k^{th} row and ℓ^{th} column crossed out. By putting the emphasis on a large lower bound for the larger problem, we rule out non-optimal tours faster.

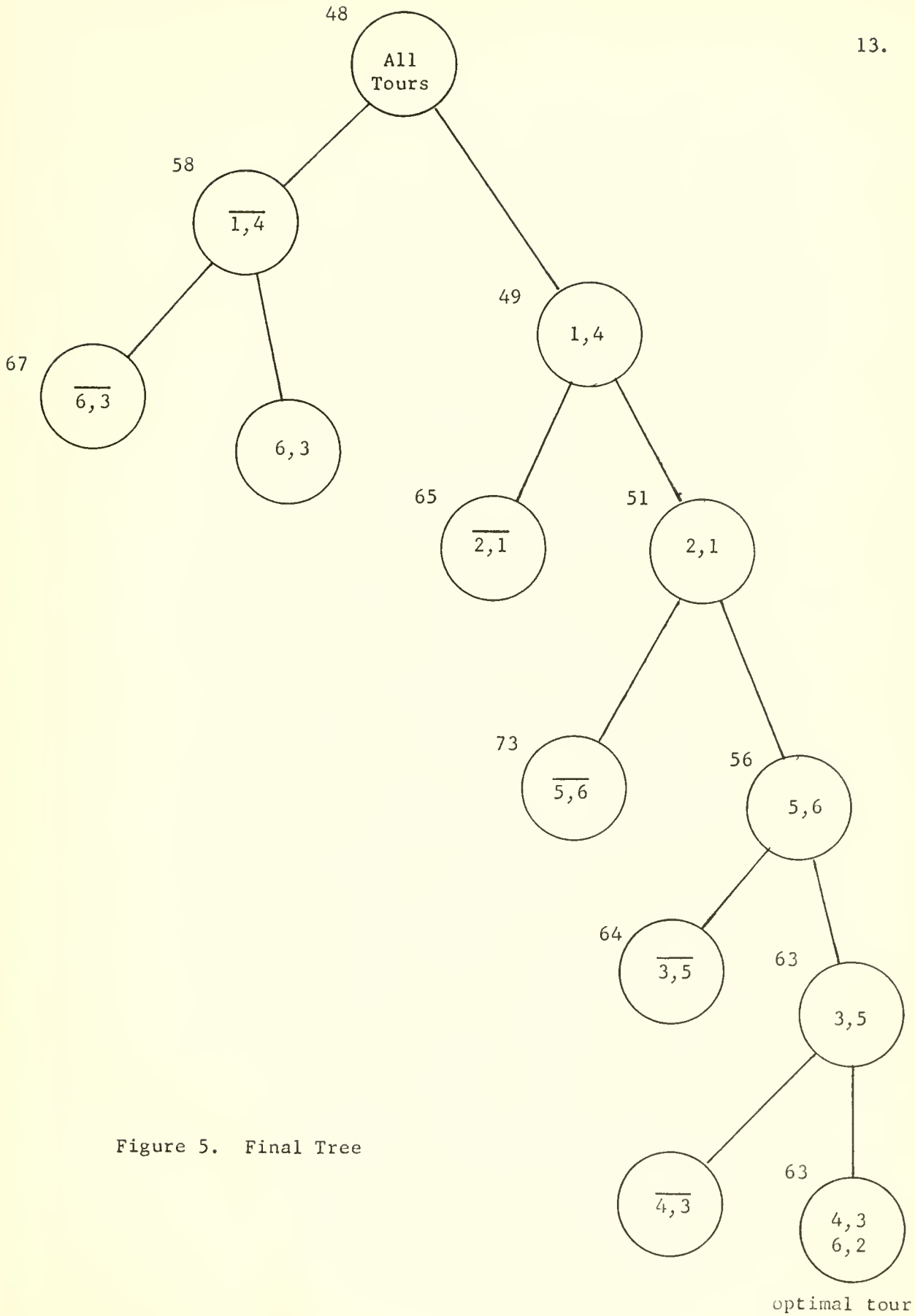


Figure 5. Final Tree

Insight into the operation of the algorithm is gained by observing that the crossing out of a row and column and the blocking of the corresponding subtour creates a new traveling salesman problem having one fewer city. Using the notation of Step 4, we can think of city $\underline{m}$ and city $\underline{p}$ as coalesced into a single city, say, $\underline{m}'$. Setting $c(m,p) = \infty$ is the same as setting $c(m',m') = \infty$. The blocking of subtours is a way of introducing the tour restrictions into what is essentially an assignment problem and is accomplished rather successfully by the algorithm.

Finally, unlike most mathematical programming algorithms, the one here has an extensive memory. It is not required that a trial solution at any stage be converted into a new and better trial solution at the next stage. A trial branch can be dropped for a moment while another branch is investigated. For this reason there is considerable room for experiment in how the next branch is chosen. On the other hand the same property leads to the ultimate demise of the computation - for $\underline{n}$ sufficiently large there are just too many branches to investigate and a small increase in $\underline{n}$ is likely to lead to a large number of new nodes that require investigation.

A number of modifications of the basic method can be proposed. For one thing, it is computationally advantageous to branch to the right until this becomes obviously unwise. Thus our program always branches on the k,ℓ node unless its lower bound exceeds or equals the cost of a known tour. As a result a few extra nodes may be examined, but usually there

will be a substantial reduction in the number of setups of Step 2a. An important consequence of this modification is that the calculation goes directly to a tour at the beginning, and if convergence is slow, usually examines many complete tours. Then, if the calculations are stopped before optimality is proven, a good tour is available. Several other embellishments of the algorithm are discussed in [12].

2. A General Flow Diagram

Although the traveling salesman illustration shows an application of the branch and bound method in detail, it will perhaps be helpful to abstract the main idea. The method breaks up the set of all feasible solutions into smaller and smaller subsets and calculates for each a lower bound on the objective function of the best solution therein. The bounds guide the partitioning of the subsets and eventually identify an optimal solution: When a subset is found that contains a single solution whose objective function is less than or equal to the lower bounds for all other subsets, that solution is optimal. The subsets of solutions are conveniently represented as the nodes on the tree and the process of partitioning as a branching of the tree; hence the name given to the method.

The convergence of the process can be assured, at least in the case where the number of solutions requiring consideration is finite, by

devising a partitioning procedure that, at worst, leads to enumeration.* As we have noted, however, the computational efficiency of the process is very dependent on the methods used to perform the partitions and calculate the bounds. The successful applications so far have exploited special features of the particular class of problems at hand.

The branch and bound idea is a fairly natural one and predates [12], although the particular name was introduced there. Eastman [2] uses the method on the traveling salesman problem but applies different branching and bounding techniques from those above. Land and Doig [11] propose the idea for mixed-integer linear programs.

Consider a general discrete-variable problem in bounded variables:

$$\min z(x_1,\ldots,x_n)$$

$$g_i(x_1,\ldots,x_n) = 0 \qquad i = 1,\ldots,m$$

$$x_j \in A_j \qquad j = 1,\ldots,n$$

where A_j is a finite set of possible values for x_j. Let

$x = (x_1,\ldots,x_n)$ = a possible solution

X = a node = a set of solutions

---

*Some writers refer to the methods reported here as forms of enumeration. This is to distinguish them from other types of algorithm. I think the term may be misleading. A dictionary definition of to enumerate is "to name one by one; specify, as in a list." [19]. In solving our 40 city problems, we averaged about 5 completely specified tours per problem. A 40 city problem has approximately 10^{46} tours. By the same definition the simplex method seems rather enumerative, since it proceeds from extreme point solution to extreme point solution with convergence guaranteed by the finite (but usually huge) number of such solutions.

$\mathbb{Y}$, $\overline{\mathbb{Y}}$ = nodes constructed by branching at $\mathbb{X}$.

$\omega(\mathbb{X})$ = a lower bound on the objective function for solutions of X, i.e., $z(x) \geq \omega(\mathbb{X})$ for $x \in \mathbb{X}$.

z^o = value of the objective function for the best solution found so far.

The branching scheme we shall present is dichotomous, with $\mathbb{X}$ branching into $\mathbb{Y}$ and $\overline{\mathbb{Y}}$ as shown in Figure 6. In some cases, however, a more natural procedure will be to create more than two nodes at a time. The extension is straightforward.

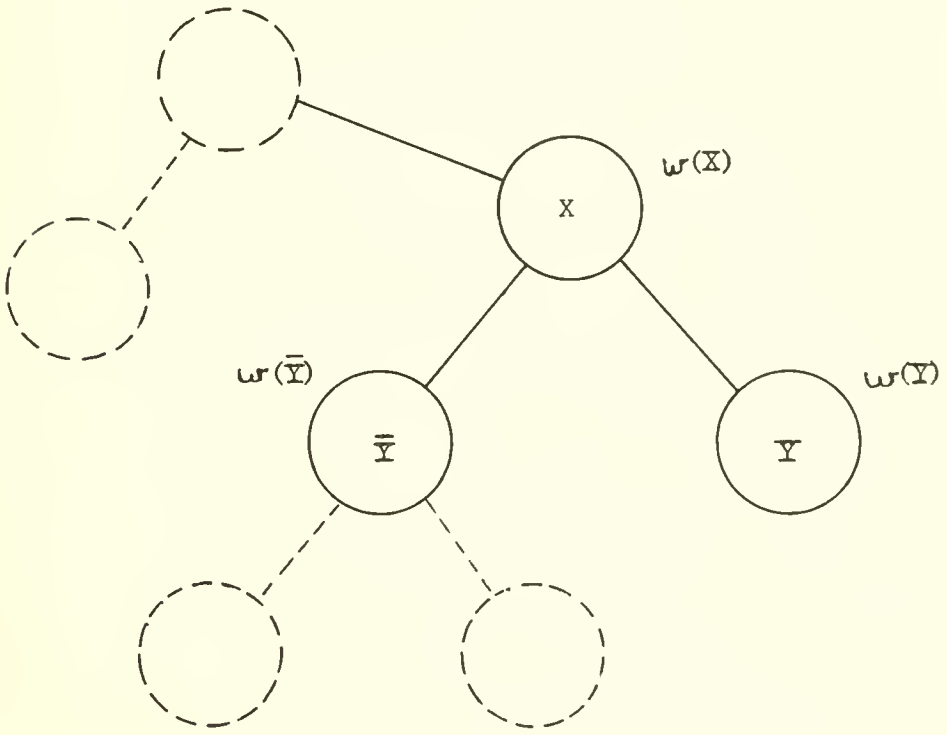


Figure 6.

Dichotomous Branching

Figure 7 shows a flow diagram for a general branch and bound algorithm. In a specific application three key ad hoc steps must be added:

(1) How to compute lower bounds.

(2) How to select x_j and assign x_j^o.

(3) How to use the constraints to eliminate infeasible solutions.

In the case of the traveling salesman problem, bounds were computed by reduction, $x_j = x_j^o$ was selected by the $\theta(k,\ell)$ calculation, and the tour constraint was used to eliminate certain infeasible solutions through setting $c(m,p) = \infty$.

3. Further Applications

Hatfield and Pierce [6] have worked on the sequencing problem. Suppose that n jobs are to be done on a machine. Let c_{ij} be the cost of changing over from job i to job j. In what order should the jobs be done to minimize the sum of the change over costs? If the n jobs form a cycle to be done repeatedly, as might be the case for models on a production line, then we have exactly a traveling salesman problem. Otherwise we can create a dummy job which costs nothing to switch to or from and without loss of generality treat the sequence as a cycle. Then we again have a traveling salesman problem. However, Hatfield and Pierce add an important practical complication, namely, due dates for the jobs. These appear as constraints that limit the number of feasible solutions. Hatifeld and Pierce also find it helpful to introduce more

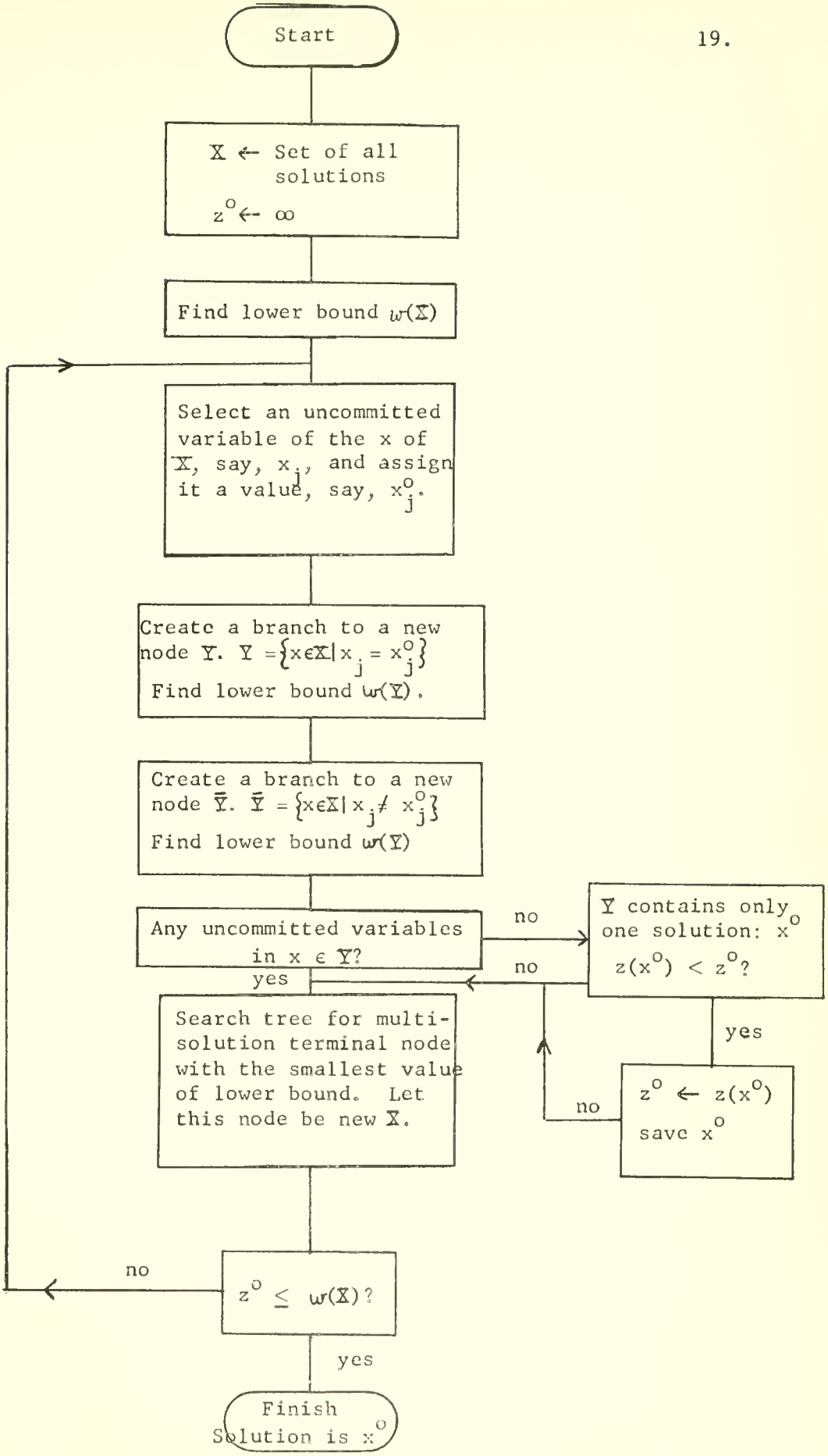


Figure 7. Flow Diagram for Discrete Variable Problem

complicated selection procedures than used above.

Lomnicki [14] and Ignall and Schrage [9] have applied branch and bound methods to certain flow-shop problems. They have achieved marked success relative to previous methods. The problem is to determine a schedule for processing n jobs on m machines. The machines are ordered and a job cannot start processing on one machine until it has been processed on the previous machine. Several objective functions are possible, for example, the time at which all jobs are finally complete (known as the makespan) or the mean completion time. Lomnicki focusses on the 3 machine case with the objective of minimizing makespan. In this case it is known that the optimal ordering of doing the jobs is the same on each machine. He uses as test problems several 6 job problems devised by Giglio and Wagner [3] for their numerical investigations of various computational approaches to the problem. Their approach for finding a guaranteed optimum was integer programming and proved quite disappointing. Some examples were not solved after 10,000 iterations on an IBM 7090. Lomnicki solved all the problems by hand.

Ignall and Schrage solved the same test problems very easily by computer with a branch and bound algorithm that is essentially the same as Lomnicki's. They solve some other 3 machine test problems as well. They further devise and demonstrate an algorithm for minimizing the mean completion time in the two machine case.

Branch and bound methods have proven helpful in work done by myself on synchronizing traffic lights [13]. The problem of setting traffic lights on an arterial street can, under certain circumstances, be formulated as a mixed integer linear program. Problems involving networks of streets can be similarly formulated. An algorithm has been devised to take advantage of the problem structure. Each light has associated with it a set of equations and variables, including one integer variable. The integer variable takes on either the value 0 or 1. Branching is done on the basis of this variable. Bounding is done by solving an r-light subproblem, the integer variables for the r lights having been specified by previous branching. The subproblem is an ordinary linear program so that the mixed integer problem is reduced to a sequence of ordinary linear problems.

4. Conclusions

We have illustrated the branch and bound idea and reviewed some of its applications. It is a technique that, if coupled with ingenuity in the devising of bounds and rules for branch selection can often produce useful algorithms for rather difficult combinatorial problems.

Bibliography

1. Dantzig, G. B., D. R. Fulkerson, and S. M. Johnson, "Solution of a Large Scale Traveling Salesman Problem," Operations Research, 2, (Nov. 1954), 393-410.

2. Eastman, W. L., "Linear Programming with Pattern Constraints," Ph.D. Dissertation, Harvard University, July 1958.

3. Giglio, R. J., and H. M. Wagner, "Approximate Solutions to the Three-Machine Scheduling Problem," Operations Research, 12, (March 1964), 305-324.

4. Gilmore, P. C. and R. E. Gomory, "Sequencing a One State Variable Machine: A Solvable Case of the Traveling Salesman Problem," Operations Research, 12, (Sept. 1964), 655-679.

5. Gonzales, R. H., "Solution of the Traveling Salesman Problem by Dynamic Programming on the Hypercube," Interim Technical Report no. 18, OR Center, MIT, 1962.

6. Hatfield, D. J. and J. F. Pierce, "On the Application of Combinatorial Programming to a Class of Single Stage Sequencing Problems," Working Paper, March 15, 1966.

7. Held, M. and R. M. Karp, "A Dynamic Programming Approach to Sequencing Problems," Journal of the Society for Industrial and Applied Mathematics, 10, (March 1962), 196-210.

8. Heller, J., "Some Numerical Experiments for an MxJ Flow Shop and its Decision Theoretic Aspects," Operations Research, 8, (March 1960), 178-184.

9. Ignall, E. and L. Schrage, "Application of the Branch and Bound Techniques to Some Flow Shop Scheduling Problems," Operations Research, 13, (May 1965), 400-412.

10. Karg, R. L. and G. L. Thompson, "A Heuristic Approach to Solving Traveling Salesman Problems," Management Science, 10, (Jan. 1964), 225-248.

11. Land, A. H. and A. Doig, "An Automatic Method of Solving Discrete Programming Problems," Econometrica, 28, (July 1960), 447-520.

12. Little, J. D. C., K. G. Murty, D. W. Sweeney, and C. Karel, "An Algorithm for the Traveling Salesman Problem," Operations Research, 11, (Nov. 1963), 972-989.

13. Little, J. D. C., "The Synchronization of Traffic Signals by Mixed-Integer Linear Programming," Operations Research, forthcoming.

14. Lomnicki, Z. A., "A Branch and Bound Algorithm for the Exact Solution of the Three-Machine Scheduling Problem," Operational Research Quarterly, 16, (March 1965), 89-100.

15. Mudrov, V. I., "A Method of Solution of the Traveling Salesman Problem by Means of Integer Linear Programming," Zhurnal Vychislennoi Matematiki i Matematicheskoi Fiziki, 3, (1963), 1137-1139.

16. Reiter, S. and G. Sherman, "Discrete Optimizing," Journal of the Society for Industrial and Applied Mathematics, 13, (Sept. 1965), 864-889.

17. Reiter, S. and G. Sherman, "Discrete Optimizing," Institute Paper no. 36, Krannert School of Industrial Administration, Purdue University, 1963.

18. Tucker, A. W., "On Directed Graphs and Integer Programs," IBM Mathematical Research Project, Technical Report, Princeton University Press, Princeton, New Jersey, 1960.

19. Webster's New World Dictionary, World Publishing Co., Cleveland, 1959.